Лабораторная работа № 6 по курсу : Операционные системы

Выполнил студент группы М8О-201Б-21 Кварацхелия Александр.

Условие

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность. Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

Задание

Топология - идеально сбалансированное дерево, команда проверки доступности узла - pingall вывести все недоступные узлы.

Метод решения

У клиента есть несколько опций: (регистрируется он в обязательном порядке) создать комнату, присоединиться к комнате, вывести статистику игрока, начать игру.

Код программы

zmq-utils.hpp

```
#ifndef ZMQ_UTILS_HPP
#define ZMQ_UTILS_HPP

#include <string>
#include <iostream>
#include <zmq.hpp>

std::string RecieveMessage(zmq::socket_t &socket);
void SendMessage(zmq::socket_t &socket, std::string &responseStr);
```

```
#endif
```

server-utils.hpp

```
#ifndef SERVER_UTILS_HPP
#define SERVER_UTILS_HPP

#include <string>
#include <sstream>
#include <vector>

#include <cstdlib>
#include <iostream>
#include <future>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <zmq.hpp>
#include <map>

struct TCmdArgs
{
    std::string command;
    std::string id;
    std::vector<int> args;
};

TCmdArgs UnpackCommand(std::string &str);

struct TWorker
{
    std::string id;
    std::future<std::string> prom;
    std::thread trd;
};

int FindById(std::vector<TWorker> &arr, std::string &id);

std::string CreateNode(std::string &id);
std::string RemoveNode(std::string &id);
void ExecNode(std::string id, std::vector<int> args, std::promise<std::string
std::string PingAll(void);
```

```
#endif
```

zmq-utils.cpp

```
#include "zmq_utils.hpp"
#include <exception>

std::string RecieveMessage(zmq::socket_t &socket)
{
    std::string result;

    try
    {
        zmq::message_t request;
        zmq::recv_result_t requestStatus = socket.recv(request, zmq::recv_fla
        result = request.to_string();
    }
    catch(std::exception &exc){
        std::cerr << exc.what() << std::endl;
    }

    return result;
}

void SendMessage(zmq::socket_t &socket, std::string &responseStr)
{
    try
    {
        zmq::message_t response(responseStr.data(), responseStr.length());
        zmq::send_result_t responseStatus = socket.send(response, zmq::send_f
    }
    catch(std::exception &exc){
        std::cerr << exc.what() << std::endl;
    }
}
```

client.cpp

```
#include "zmq_utils.hpp"

#include <string>
#include <sstream>
#include <iostream>
#include <stdexcept>
```

```
#include <zmq.hpp>

int main (int argc, char const *argv[])
{
    zmq::context_t context;
    zmq::socket_t socket(context, zmq::socket_type::pair);

    const std::string address = "tcp://localhost:4040";
    socket.connect(address);

    std::string str;

    while (std::getline(std::cin, str))
    {
        if (str.length() > 0)
        {
            try
            {
                SendMessage(socket, str);
            }
            catch (std::exception &exc){
                std::cerr << exc.what() << std::endl;
            }
        }
    }

    str = "END_OF_INPUT";
    SendMessage(socket, str);

    while (true)
    {
        zmq::message_t response;
        zmq::recv_result_t responseStatus = socket.recv(response, zmq::recv_f

        if (!response.to_string().compare("END_OF_INPUT")){
            break;
        }

        std::cout << response.to_string() << std::endl;
    }

    socket.disconnect(address);
```

```
    socket.close();
    context.close();

    return 0;
}
```

server.cpp

```
#include "server_utils.hpp"
#include "zmq_utils.hpp"

#include <queue>
#include <thread>
#include <future>

int main (int argc, char const *argv[])
{
    zmq::context_t context;
    zmq::socket_t socket(context, zmq::socket_type::pair);

    std::string address = "tcp://*:4040";
    socket.bind(address);

    std::vector<TWorker> workers;
    std::queue<std::string> responseQueue;

    while (true)
    {
        std::string requestStr = RecieveMessage(socket);
        TCmdArgs arguments = UnpackCommand(requestStr);

        if (!arguments.command.compare("create")){
            responseQueue.push(CreateNode(arguments.id));
        }

        if (!arguments.command.compare("exec"))
        {
            TWorker worker;

            std::promise<std::string> workerProm;
            std::future<std::string> futureValue = workerProm.get_future();
            std::thread trd(&ExecNode, arguments.id, arguments.args, std::mov
```

```
            worker.id = arguments.id;
            worker.prom = std::move(futureValue);
            worker.trd = std::move(trd);

            workers.push_back(std::move(worker));
        }

        if (!arguments.command.compare("remove"))
        {
            int idx = FindById(workers, arguments.id);

            if (idx != -1)
            {
                workers[idx].trd.join();
                responseQueue.push(workers[idx].prom.get());

                workers.erase(workers.begin() + idx);
            }

            responseQueue.push(RemoveNode(arguments.id));
        }

        if (!arguments.command.compare("pingall")){
            responseQueue.push(PingAll());
        }

        if (!requestStr.compare("END_OF_INPUT"))
        {
            for (TWorker &elem : workers)
            {
                if (elem.trd.joinable()){
                    elem.trd.join();
                }

                responseQueue.push(elem.prom.get());
            }

            break;
        }
    }

    while (!responseQueue.empty())
```

```
    {
        zmq::message_t response(responseQueue.front().data(), responseQueue.f
        zmq::send_result_t responseStatus = socket.send(response, zmq::send_f

        responseQueue.pop();
    }

    std::string terminate = "END_OF_INPUT";
    zmq::message_t response(terminate.data(), terminate.length());
    zmq::send_result_t responseStatus = socket.send(response, zmq::send_flags

    socket.close();
    context.close();

    return 0;
}
```

Выводы

В процессе выполнения лабораторной работы были приобретены навыки практического применения очередей сообщений, потоков и процессов.